SANDIA REPORT

SAND2004-XXXX
Internal Use Only
Printed May 2004

ML 3.1 Developer’s Guide

Marzio Sala, Jonathan J. Hu, Ray S. Tuminaro

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories

Telephone:	(865) 576-8401
Facsimile:	(865) 576-5728
E-Mail:	reports@adonis.osti.gov
Online ordering:	http://www.doe.gov/bridge

Telephone:	(800) 553-6847
Facsimile:	(703) 605-6900
E-Mail:	orders@ntis.fedworld.gov
Online ordering:	http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2004-XXXX
Internal Use Only
Printed May 2004

ML 3.1 Developer's Guide

Marzio Sala
Computational Math & Algorithms
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-1110

Jonathan J. Hu and Ray S. Tuminaro
Computational Math & Algorithms
Sandia National Laboratories
P.O. Box 0969, MS 9159
Livermore, CA 94551-0969

Abstract

ML is a multigrid preconditioning package intended to solve linear systems of equations $Ax = b$ where A is a user supplied $n \times n$ sparse matrix, b is a user supplied vector of length n and x is a vector of length n to be computed. ML should be used on large sparse linear systems arising from partial differential equation (PDE) discretizations. For an overview of ML , we refer to the ML users' guide.

This guide is intended for anyone who will be adding to or modifying the ML source code. This document contains suggested practices, naming conventions, and autoconf/automake hints. We don't intend for this document to be a set of hard-and-fast rules, but rather *suggested* and *encouraged* practices. These guidelines are intended to be complimentary to policies established in the Trilinos Developers Guide [HWH03].

Acknowledgments

The authors would like to acknowledge the support of the ASCI and LDRD programs that funded the development of ML.

ML 3.1 Developer's Guide

Contents

Part I
ML and Related Tools

```
int i;main()for(;i--<i;++i)--i;"];
read(i+++,"hello, world!",++i++));
read(j,i,p)write(j/p+p,i---j,i/i);
```
–Obfuscated C Code. Author requested anonymity.

1 Introduction

ML development was started in 1997 by Ray Tuminaro and Charles Tong. Currently, there are several full- and part-time developers. The kernel of ML is written in ANSI C, and there is a rich C++ interface for Trilinos users and developers.

ML can be customized to run geometric and algebraic multigrid; it can solve a scalar or a vector equation (with constant number of equations per grid node), and it can solve a form of Maxwell's equations. For a general introduction to ML and its applications, we refer to the Users Guide [SHT04], and to the ML web site, http://software.sandia.gov/ml.

1.1 Extensibility

ML is designed to be *flexible.* Developers can easily add, for example,

- new aggregation schemes;
- new smoothers;
- new coarse solvers;
- new multigrid cycles.

However, developers should keep the following goals in mind:

1. Code should be robust and error free;
2. Code should be easy to use and understand;
3. Code should be easy to maintain.

This guide furnished guides, suggestions, and guidelines to help present and future ML developers.

Although all sections of this guide will be useful to most developers, it is worth mentioning that this guide supports three types of development activities:

1. Configuration and building: how to manage the autotools, how to add a new file/example/test;
2. Documentation: how to write Doxygen comments. This guide is not an introduction to Doxygen, but we give some simple hints to generate efficient Doxygen documentation;
3. Coding conventions: how to write code in ML .

Remark 1 *As a note, we recall that guidelines for* ML *coding style and other suggestions are just that, guidelines and suggestions. They are not intended to be obeyed zealously. In fact, much of* ML *does not strictly adhere to these guidelines because most of the* ML *code was developed when this developer's guide did not yet exist.*

2 How To Use This Guide

The goal of this document is to guide new ML developers in:

- getting started (in Section 4);
- using available development tools: Bonsai, Bugzilla, Mailman, and the cvs repository (in Section 5);
- modifying how ML is configured and built (in Section 6);

- adding a new test to the ML repository (in Section 7);
- writing Doxygen documentation (in Section 8);
- defining some suggested practices to write code for ML (in Section 11);
- better understanding the ML structures (in Section 12);

Remark 2 *The aim of the following section is to be useful in that part of code development that has proved to be painful in the past. Unfortunately, only a limited part of* ML *'s capabilities will be covered in this document. Developers are encouraged to*

1. *keep this guide up to date after each relevant change;*
2. *add his own experience;*
3. *extend the manuscript whenever important topics arise.*

3 Notational Conventions

In this guide, we show typed commands in this font:

```
% a_really_long_command
```

The character `%` indicates any shell prompt[1]. Function names are shown as `ML_Gen_Solver`. Names of packages or libraries as reported in small caps, as EPETRA. Mathematical entities are shown in italics.

4 Getting Started

ML can be obtained in different ways. Here, we suppose that the developer has access to the CVS repository on `software.sandia.gov`. In addition, the user account must be in the `trilinos` and `cvs` group on this computer. To request an account, send a note to `trilinos-help@software.sandia.gov`. The following variables must be defined (and exported):

```
CVSROOT=:ext:your_user_name@software.sandia.gov:/space/CVS
CVS_RHS=ssh
```

(Replace `your_user_name` with your login name.) To check out a working copy of ML in the current directory, type

```
cvs checkout ml
```

For a more detailed description of CVS commands, we refer to the Trilinos Developers Guide [HWH03], and to the GNU CVS home.

Please refer to the ML Users Guide for guidance in configuring ML , We suggest creating a simple script that contains all the parameters for `configure`. Be sure to use continuation characters ('\') properly. The characters should be at the end of every line, except the last line, and should not be followed by any space. Recall that autoconf cannot detect spelling mistakes in configure invocation scripts.

Remark 3 *Other tips for making the configure and build more efficient can be found in [HWH03, Section 2.6].*

[1]For simplicity, commands are shown as they would be issued in a Linux or Unix environment. Note, however, that ML has and can be built successfully in a Windows environment.

5 Related Tools

5.1 Bonsai

Bonsai is a viewer for the CVS repository that runs via a web browser. Some of its capabilities include

1. file browsing, with or without ”blame” annotation
2. differencing of file versions
3. repository searches based on file or directory names
4. determining CVS changes based on date or time

Bonsai is located at http://software.sandia.gov/bonsai.

5.2 Bugzilla

Users and developers are encouraged to use Bugzilla, to report configuration problems, bugs, suggest enhancements, or request new features. Bugzilla can be found on the web at

```
http://software.sandia.gov/bugzilla
```

If reporting a configuration problem or a bug, please attach the configure script that has been used, and the compilation and/or run-time error.

Remark 4 *When checking in a fix that addresses a bug reported in bugzilla, include the bug number. Also include a short description of the fix.*

5.3 Mailing Lists

Any substantive developer discussion should take place on the ML developer’s list,

```
ml-developers@software.sandia.gov
```

If the discussion is off-line, it’s entirely appropriate to email a summary of the discussion to the list. The list archives the discussion. This can be helpful to the developers. Moreover, this discussion can be used as documentation during external reviews.

Part II
Configuration and Building

"C combines the power of assembler with the portability of assembler." – Anonymous

6 Modifying How ML is Configured and Built

ML is built using the GNU tools autoconf and automake [Frea, Freb].

As a small example of how to add a new configure option, let us assume that the current working director is `Trilinos/packages/ml`. The file configure.ac contains all of the configure line option definitions (e.g., `--enable-ml_flops`). Suppose that we want to add the configure option "`--enable-ml_foo`". (Note that `ml_foo` has an underscore and *not* a dash.)

We first add the following line to `configure.ac`:

```
TAC_ARG_ENABLE_OPTION(ml_foo, [This enables the foo option.], ML_FOO, no)
```

We then run bootstrap:

```
% ./bootstrap
... some output here ...
```

Among other things, bootstrapping will modify `src/ml_config.h.in` and create a new macro, `HAVE_ML_FOO`. When ML is configured, the file `ml_config.h` is created. If the option `--enable-ml_foo` was supplied on the command line, then `HAVE_ML_FOO` will be defined in `ml_config.h`.

We could use this macro directly in ML . Suppose, however, the macro `ML_FOO` is already used heavily in ML , and we don't want to change the ml source. ML has an include file, `src/Include/ml_common.h`, that is included in every ML source file. We add the following to `ml_common.h`:

```
#ifdef HAVE_ML_FOO
#define ML_FOO
#endif
```

and voila! Adding `--enable-ml_foo` on the configure line will now define the macro `ML_FOO` inside the ml source.

7 How and Why to Add a New Test

ML has a test suite that is automatically executed every night on a variety of platforms. This suite verifies that important part of the code execute correctly, and that latest changes do not break the existing code. For more details on the test harness, we refer to [HWH03, Section 3.3].

There are two ways of adding a new test:

1. Method 1: Adding a separate executable just for testing.

   (a) Create a new subdirectory in `<ml-dir>/test` (for example, `new-test`).

   (b) Put the test source code and the corresponding makefile in `new-test`.

   (c) Add `new-test` to the script file(s), that is located in one or all of the following:

   ```
   <ml-dir>/test/scripts/daily/mpi
   <ml-dir>/test/scripts/daily/serial
   <ml-dir>/test/scripts/weekly/mpi
   <ml-dir>/test/scripts/weekly/serial
   ```

   (Currently, ML has daily test only.) `new-test` should be added to the `foreach` block.

   (d) Modify `<ml-dir>/configure.ac`, by adding `new-test` to the list contained in `AC_CONFIG_FILES`.

   (e) Run `bootstrap`.

2. Method 2: Using an example from ml/examples for testing.
   (a) Create a new subdirectory in `<ml-dir>/test` (for example, `new-test`).
   (b) Create an appropriate `Makefile.am` in `new-test`. We suggest copying and modifying the `Makefile.am` from `ml/test/2d_Poisson`.
   (c) Create an appropriate driver script in `new-test` by copying and modifying the `2d_Poisson.csh` script from `ml/test/2d_Poisson`. You should only need to change the value of the executable name at the top of the script.
   (d) Append `new-test` to the variable `TEST_SUBDIRS` in the script file `Test_MLExamples` located in one or all of the following:

   ```
   <ml-dir>/test/scripts/daily/mpi
   <ml-dir>/test/scripts/daily/serial
   <ml-dir>/test/scripts/weekly/mpi
   <ml-dir>/test/scripts/weekly/serial
   ```

   (Currently, ML only has daily tests.)
   (e) Modify `ml/configure.ac` by adding `new-test` to the list contained in `AC_CONFIG_FILES`. (This is near the bottom of `configure.ac`.)
   (f) Run `bootstrap`.

A new test should be added to the test harness suite when a new feature has been included in ML .

8 How to Write Doxygen Documentation

ML uses doxygen for the comments in C++ code. As Doxygen generally does not produce good output for C code, the ML C files are not Most Doxygen-complaint. (However, the most important ML structures have Doxygen comments.)

This section gives some general guidelines about how to write Doxygen documentation. First, a comment on writing comments: you want your comments to tell *what your* code does, not *how*. Remember also that comments are good, but there is also a danger of over-commenting. You can make small comments to note or warn about something particularly clever (or ugly), but try to avoid excess. Instead, put the comments at the head of the function, telling people what it does, and possibly *why* it does it. Ideally, The best way to write comment would be to write the code so that the working is obvious, and it's a waste of time to explain badly written code.

In C++ code, developers should adopt Doxygen-style comments for every class. These comments will be included in the header file. This is where the interface is, and where people usually look for help.

It is suggested to start a new file with a small Doxygen comment of type:

```
/*!
 *  \file <file name>
 *
 *  \brief <Brief description of file content>
 *
 *  \author <Author Name>
 *
 *  \date <Creation and last modification>
 *
 */
```

Functions/methods can be commented in Doxygen style as

```
/*! Here a brief description of the function */
/**
 * Search a string in a buffer.
 *
 * \param buffer (In) the buffer in which to search.
 * \param string (In) the string to look for.
 * \return the index of the first occurrence.
 */
int ML_find_string(Buffer& buffer, String& string);
```

This allows the automatic generation of HTML documentation.

Remark 5 *Mark every bug and/or potential problem in the code with a comment starting with* `FIXME:`*. This makes it very easy to locate such problems with tools like grep. Also, some editors (like* `vim`*) automatically highlight the FIXME keyword.*

9 Exporting ML 's Dependencies to Other Packages

ML 's dependencies are exported to other packages via the file `ml/Makefile.export.in`. When ML is configured, autoconf produces the file `Makefile.export`, which defines the variable `ML_EXPORT_LIBS`. Packages that depend on ML should use `ML_EXPORT_LIBS` in their configure and build process to ensure their link lines are correct.

If a dependency on package `XX` is introduced during ML development, this dependency should be added to `ml/Makefile.export.in` as follows.

1. Modify the definition of `ML_EXPORT_LIBS` by appending

   ```
    -L$(XX_BUILD_DIRECTORY)/src  $(XX_LIBS)
   ```

2. Give the location of `XX`'s build directory:

   ```
   XX_BUILD_DIRECTORY = $(ML_BUILD_DIRECTORY)/../XX
   ```

3. Give the name of the XX library:

   ```
   HAVE_ML_XX = @HAVE_ML_XX@
   ifeq ($(HAVE_ML_XX),true)
   XX_LIBS = libXX.a
   else
   XX_LIBS =
   endif
   ```

   Note that `XX_LIBS` is defined only if ML has been configured with `XX` enabled.

10 FAQ

1. **Question:** The **./configure** command fails with the following error:

```
checking for Fortran 77 libraries...
checking for dummy main to link with Fortran 77 libraries... unknown
configure: error: linking to Fortran libraries from C fails
See ‘config.log’ for more details.
```

Answer: The most likely problem is an incorrect configure line option. Check that all of the library and include locations that you've specified are correct. Look in `config.log` to find the exact error.

2. **Question:** I am building with LAM under RH9 Linux, and `configure` complains that it cannot find `mpi++.h`.
   Answer: Add `-DLAM_BUILDING` to your CXX parameters; for instance,

   ```
   ../configure --with-cxxflags="-DLAM_BUILDING"
   ```

3. **Question:** I'm getting warnings about non-modifiable left hand sides when using `ML_free`.
   Answer: `ML_free` is a macro. No casting of the argument is necessary or even correct. The argument to `ML_free` is used within the macro source on the left hand side of a logical check.

   Good: ML_free(i);

   Bad: ML_free((void *) i); (does not compile on SGI, for instance)

4. **Question: Are C++ style comments, i.e, `//`, ok to use in ML?**
   Answer: C++ style comments are fine to use in C++ files. You must use C style comments, i.e., `/* */`, in `*.c` files. Otherwise, the code will not compile on either Solaris machines or on Janus.

5. **Can I use C99?**
   Answer: Most compilers do not support C99 standard. Only C89 code should be included in the ML distribution. The rational is that many platforms (like ASCI-Red, for instance) still have very old compilers, that are not C99-compliant.

Part III

Coding Practices

When the code and the comments disagree, both are probably wrong.

11 Suggested Practices for C++ code

ML is written in both C and C++. The C++ programming language differs substantially from the C programming language. In terms of usage, C is more like Pascal than it is like C++.

Therefore, it is important to adopt good coding habits, one for C, and another for C++. A good style guide can enhance the quality of the code that we write. This Section tries to present a standard set of methods for achieving that end for C++ code.

It is, however, the end itself that is important. Deviations from this standard style are acceptable if they enhance readability and code maintainability. Major deviations require a explanatory comment at each point of departure so that later readers will know that you didn't make a mistake, but purposefully are doing a local variation for a good cause.

Further suggestions on "how to write good C/C++ code" can be found, for instance, in the NOX guide, and the Epetra Developers Coding Guidelines [MAH03].

11.1 General

- Any code that links to Trilinos, and any Trilinos file must define `HAVE_CONFIG_H`.
- Although there is no maximum length requirement for source files, long files are cumbersome to deal with.
- Lines longer than 80 columns should be avoided. Use C/C++'s string concatenation to avoid unwieldy string literals and break long statements onto multiple lines.

  ```
  char *s1 = "hello\n"
             "world\n";                    // s1 is exactly the same as s2,
  char *s2 = "hello\nworld\n";
  ```

  The line length limit is related to the fact that many printers and terminals are limited to an 80 character line length. Source code that has longer lines will cause either line wrapping or truncation on these devices. Both of these behaviors result in code that is hard to read.
- No `#pragma` directive should be used. `#pragma` directives are, by definition, non-standard, and can cause unexpected behavior when compiled on other systems. On another system, a `#pragma` might even have the opposite meaning of the intended one.
- Macros are seldom necessary in C++. The construct `#define NAME` value should never be used. Use a `const` or `enum` instead, because the debugger can deal with them symbolically, while it can't with a `#define`, and their scope is controlled and they only occupy a particular namespace, while `#define` symbols apply everywhere except inside strings.

  Macros in C are frequently used to define "maximum" sizes for things. This results in data structures that impose arbitrary size restrictions on their usage, a particularly insidious source of bugs. Try not to carry forward this limitation into C++.
- When incrementally modifying existing code, follow the style of the code you are modifying, not your favorite style. Nothing is harder to read than code where the personal style changes from line to line.
- Don't use global data. Consider using file- or class-static data members instead. (However, the use of static data should be minimized if not avoided.)
- A char may be unsigned or signed. You can't assume either. Thus, only use (unmodified) char if you don't care about sign extension and can live with values in the range of 0-127.

- Always provide the return type of a function explicitly, The value being returned should be enclosed in parenthesis.

```
  return i;  // No!
  return(i); // Yes
```

- Functions with a return type of void should use an ”empty” return determent.

```
  void foo() {
     ...
     return;
  }
```

- Functions that don’t take any parameter should use an empty parameter list, and not say void.
- Always define a pointer when you declare it. Either set it equal to an address in memory, or set it equal to zero. If you don’t define a pointer as you declare it, you will never know if it will be accessed before you assign to it. This goes both for local variables, and for class members in constructors.
- Use zero (0) instead of `NULL` in C++ code.
- Always include a default case in a `switch` statement, or an `else` in a sequence of `if-else if`’s.
- Do not use spaces around ‘`.`’ or ‘`->`’, or between unary operators and operands.
- Always provide a space on both sides of ‘`=`’ signs and all virginal operators.
- Use parenthesis to make the code readable.
- The block of any `if` statement should always follow on a separate line.

```
if (/*Something*/) i++; // No!

if (/*Something*/)      // Yes!
  i++;
```

- Operators should have a space on both sides of them. (Exception if the `*` and & deferencing operators.). This makes it easier to distinguish which usage is intended.

```
int* a   // defining a pointer to int
a * b    // multiplying two variables
*a       // deferencing a pointer
```

11.2 File Naming Conversions

- C and C++ header files end in `.h`;
- C source files end in `.c`, and C++ source files end in `.cpp`;
- All file (header and source, for library and examples) begin with `ml_`;
- For C++ files, the name of the files should correspond to the name of the class they define.

11.3 Include File Structure

- Every include file must contain a mechanism that prevents multiple inclusions of the file. For example, the following should follow the header information for the file `ml_foo.h`:

  ```
  #ifndef ML_FOO_H
  #define ML_FOO_H

  ...body of include file goes here

  #endif
  ```

- Definition of classes that are only accessed via pointers (`*`) or references (`&`) should be declared using forward declarations, and not by including the header files.

11.4 Naming Conventions

- All ML functions should begin with `ML_`;
- All ML -Epetra C++ functions and classes should be in the namespace `ML_Epetra`;
- All ML functions and class names should begin with an uppercase letter;
- All `Get` or `Set` functions should be as follows: `ML_Get_PrintLevel()`;
- All ML class data members should end with an underscore (e.g. `int NumLevel_`). No other variable names should ever end with an underscore;
- Do not use identifiers that begin with one or two underscores;
- Accessor method should have the same name as the attribute they access, without the underscore:

  ```
    int someVar_;
    int SomeVar() { return someVar_);
  ```

- Variables used for loops counters should be names `i,j,k`, etc. in that order.
- For C++ code, C-style casts should never be used. User `static_cast`, `reinterprest_cast`, and `const_cast` instead.
- `const_cast` should be avoided as much as possible. When you need to modify an object that is logically const but not bitwise const, use the `mutable` keyword instead.
- Member definition in constructors: Member definition should be formatted as follows, each on their own line, with the colon preceding the first one, a comma following all but the last one, and the opening curl brace of the function body on a new line.

  ```
    MLClass::MLClass(int foo)
    : SomeMemberVar(0),
    SomeOtherVar(foo+1)
    {
      ...
    }
  ```

- Declare only one variable per line.

  ```
  int i,j; // No!
  int i;
  int j;
  ```

  This is mainly to avoid confusion resulting from mixing `int` and `int *` declarations, and also to give room for additional comments (when required).

- The inclusion of every non-C++ header file must be surrounded by the extern "C" construct.
- Function calls that are intended to be called from C that take input-only struct arguments may wish to use pointers, since C does not have references. Such pointers must, of course, be declared const.
- The public, protected and private section of a class are to be declared in that order;
- Friend class declarations should immediately precede the private section, to emphasize that they too can access those members;
- The order functions are listed in the `.c.` or `.cpp` file should match the order they are listed in the class declaration in the `.h` file.

11.5 Indentation

- (Loose) convention is to put the opening brace last on the line, and put the closing brace first:

  ```
  if (x is true) {
    ... function body ...
  }
  ```

  However, there is one special case, namely functions: they have the opening brace at the beginning of the next line, thus:

  ```
  int function(int x)
  {
    body of function
  }
  ```

  Note that the closing brace is empty on a line of its own, except in the cases where it is followed by a continuation of the same statement, ie a "while" in a do-statement or an "else" in an if-statement, like this:

  ```
  do {
    body of do-loop
  } while (condition);
  ```

- The characters '`*`' and '&' should be written in the following way:

  ```
  int* pointer;
  double& ref;
  ```

  Instead of saying the `*i` is of type `int`, say that `i` is of type `int*`.

- An else statement following an if should begin on the line following the i's closing brace.

  ```
  if (x == y) {
    ...
  }
  else if (x > y) {
    ...
  }
  else {
    ...
  }
  ```

- Do not put spaces between function names and the parenthesis that starts the list of arguments. It makes it less obvious that the list of parameters belongs to that function call. Also, leave one space between parameters, after the comma, but don't leave any space before the first parameter, after the last one, or before a comma. If the function has no parameters, don't leave a space between parenthesis.

  ```
    void foo( );       // No!
    void foo (1, 2);   // No!
    void foo( 1, 2 );  // No!
    void foo( 1 , 2 ); // No!

    void foo();     // Yes
    void foo(1, 2); // Yes
  ```

- When defining functions, the leading parenthesis and the first argument (if any) are to be written on the same line as the function name. If space permits, other arguments and the closing parenthesis may also be written onthe same line as the function name. Otherwise, each additional argument is to be written on a separate line (with the closing parenthesis directly after the last argument).

  ```
    ML_function(int FirstParameter,     // No!
      int SecondParamete, int ThirdParameter)
    {
      ..
    }

    ML_function(                        // No!
      int FirstParameter,
      int SecondParamete,
      int ThirdParameter
    ) {
      ..
    }

    ML_function(int FirstParameter, int SecondParameter,  // Yes
                int ThirdParameter)
    {
      ...
    }
  ```

- `cin/cout/cerr`-like indentation should look like this:

```
cout << "Problem:\t" << problem.name()
     << "Solution:\t" << problem.solution()
     << endl;
```

- If you split an expression into multiple lines, split it after an operator, not before it:

```
if (condition_number_one &&
    condition_number_two &&
    condition_number_three)
```

11.6 Portability

Probably, the only way to write really portable code, is to test extensively on all the desired (and available) architectures. Using `g++`, the following flags can be useful to detect non-ANSI features: `-ansi -pedantic -Wall`. Flag `-ftrapv` can cause ML to crash, as some random functions generate an overflow. Flag `-Weffc++` can be very helpful, but recall that system files can produce a lot of warnings with this flag.

Part IV

Extending ML

`ML_CHK_ERR(ierr)`	If `ierr != 0`, this macro prints out an error message, and returns `ierr`.
`ML_CHK_ERRV(ierr)`	If `ierr != 0`, this macro prints out an error message, and returns void.
`ML_RETURN(ierr)`	If `ierr != 0`, this macro prints out an error message. This macro always returns `ierr`.
`ML_EXIT(ierr)`	If `ierr != 0`, this macro prints out an error message. This macro always exits.

Table 1. C++ return and exit macros.

12 Main Structures of ML

12.1 Managing Memory

ML has macros that wrap the system calls to allocate and free memory. "ML_allocate" should be used instead of "malloc". "ML_free" should be used instead of "free".

12.2 Error Handling

The code should always check return values of functions for errors and, whenever possible, try to recover from errors, by catching an integer return value[2]. The use of `assert()` calls extensively to check the invariants in your code. This will dramatically decrease your debugging time by catching inconsistencies early. See also Table 1.

12.3 Output

ML uses the concept of *print level*. Each output sentence has a value from - to 10 (10 being verbose), and it will be printed out only if the current print level is below this value. Each print statement should be preceded by a `ML_Get_PrintLevel()`. Standard output should be sent to `stdout` (C) or `cout` (C++). Warnings and errors should be sent to `stderr` (C) and `cerr` (C++).

12.4 How Getrow() Works

The following simple code can be used to get all local rows of an `ML_Operator`, here called `Amat`.

```
int allocated = 1;
int* colInd = new int[allocated];
double* colVal = new double[allocated];
int NumNonzeros;
int ierr;

for (int i = 0 ; i < Amat->outvec_leng ; ++i)
{
  ierr = ML_Operator_Getrow(Amat,1,&i,allocated,colInd,colVal,&NumNonzeros);

  if (ierr == 0) {
```

[2]Currently ML does not support `try/catch` blocks.

```
    do {
      delete [] colInd;
      delete [] colVal;
      allocated *= 2;
      colInd = new int[allocated];
      colVal = new double[allocated];
      ierr = ML_Operator_Getrow(Amat,1,&i,allocated,colInd,colVal,&NumNonzeros);
    } while (ierr == 0);
  }
  // .. do something with the row elements
}
delete [] colInd;
delete [] colVal;
```

The error code `ierr` is zero if the `allocated` space was enough to copy on vectors `colInd` and `colVal` the nonzero elements and their (local) column numberfor row `i`. If `ierr==0`, then the user can reallocate `colInd` and `colVal`, using a larger chunk of memory, and recall the getrow() function.

12.5 Apply the Smoother to a Vector

The following fragment of code can be used to apply the pre/post smoother defined for level `ilevel` to a vector `rhs`. The starting solution is defined in vector `rhs`. We suppose that the ML hierarchy has already been filled with the appropriate structures.

```
ML_Smoother * ptr = ((ml_->SingleLevel[ilevel]).pre_smoother);

int length = ml_->Amat[ilevel].outvec_leng;
double sol[length];
double rhs[lenght];
// ... here define the elements for sol and rhs
// the number of smoother applications is ptr->ntimes;

ML_Smoother_Apply(ptr, length,
  tmp_sol, length, rhs, ML_NONZERO);
```

12.6 Print an ML_Operator in a File

The following simple fragment of code can be used to dump an `ML_Operator` to a file, say `my_operator`.

```
// ml is an ML_struct, already filled with the appropriate operators
char name[] = "my_operator";
ML_Operator_Print(&(ml->Amat[LevelID_[i]]), name);
```

The code works in serial and parallel. Serial runs will create a file called `my_operatator.serial`; this file can be easily read by MATLAB, where a sparse matrix can be created using `spconvert()`. Parallel runs will create a file for each process. For parallel runs, one may find useful the following function:

```
int ML_Operator_Print_UsingGlobalOrdering(ML_Operator *matrix,
                                          const char label[],
                                          int *global_row_ordering,
                                          int *global_col_ordering)
```

This function prints a `ML_Operator` into MATLAB format. Only one file is generated using global ordering. In input, `matrix` is an `ML_Operator`, distributed among the proceses If the matrix is rectangular, the user should pass in both a global row ordering and a global column numbering. The matrix will be written in MATLAB (i,j,k) format to file `label.m`. The last two parameters can be se to NULL.

12.7 Timing

The following variables can be used to track timing:

- `ML_TIMING`;
- `ML_TIMING_DETAILED`.

12.8 Checking Memory Usage

Probably, the easiest way to check the memory usage (under Linux) is to use `valgrind`, like for instance

```
% valgrind --leak-check=yes --show-reachable=yes ./ml_example.exe
```

This also works in parallel:

```
% mpirun -np 4 valgrind --leak-check=yes --show-reachable=yes ./ml_example.exe
```

(but it is slower, and can produce a lot of output.)

Another way is to compile ML with the flag `ML_MEM_CHECK`. Then, the user can insert a call to `ML_print_it()`, to print out all the ML allocagted memory that is still to be deleted. This check can be expensive for large runs.

12.9 Attaching a Debugger to ML

To debug ML on a serial computer (that is, using only one processor), one can simply type somthing like

```
% gdb ./mlguide.exe
GNU gdb Red Hat Linux (5.3post-0.20021129.18rh)
Copyright 2003 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux-gnu".
(gdb) r
... output of the code ...
```

Unfortunatly, most debuggers (like `gdb`) cannot directly run parallel applications. In this case, one can still attach to the desired process (or possibly to more than one), in a very simple way:

1. In the part of the code where one wish to start debugging, call the following function:

   ```
   ML_PauseForDebugger(Comm);
   ```

   where `Comm` is an `ML_Comm` structure;

2. Define the environmental variable `ML_BREAK_FOR_DEBUGGER`, for instance (using BASH)

   ```
   $ export ML_BREAK_FOR_DEBUGGER=1
   ```

3. Alternatively, if using class MultiLevelPreconditioner, create a file called `ML_debug_now` in the working directory.

4. Run the code as required,

```
$ mpirun -np 2 ./mlguide.exe
Host and Process Ids for tasks
Host: s850675.sandia.gov   PID: 4153
Host: s850675.sandia.gov   PID: 4154

** Pausing because environment variable ML_BREAK_FOR_DEBUGGER has been set.
**
** You may now attach debugger to the processes listed above.
**
** Enter a character to continue >
```

   The code prints out the process ID of each ML's process. After attaching, the user proceed by inserting a char value.

12.10 Structure ML_Comm

The structure `ML_Comm` is defined as follows:

```
typedef struct ML_Comm_Struct
{
   int      ML_id;
   int      ML_mypid;
   int      ML_nprocs;
   USR_COMM USR_comm;
   int      (*USR_sendbytes)(void*,unsigned int,int,int,USR_COMM);
   int      (*USR_irecvbytes)(void*,unsigned int,int*,int*,USR_COMM,USR_REQ*);
   int      (*USR_waitbytes)(void*,unsigned int,int*,int*,USR_COMM,USR_REQ*);
   void     (*USR_cheapwaitbytes)(void*,unsigned int,int*,int*,USR_COMM,USR_REQ*);
   USR_ERRHANDLER *USR_errhandler;

} ML_Comm;
```

To get the process ID, simply use `comm->ML_mypid`. To get the MPI communicator, one can proceed as follows:

```
int orig_comm;
orig_comm = comm->USR_comm;
// now orig_comm can be used as MPI_COMM_WORLD
```

12.11 Sparsity Pattern of an `ML_Operator`

Function `ML_Operator_PrintSparsity()` can be used to visualize the sparsity pattern of an ML_Operator. The function, that can be used for both serial and parallel run, has the following syntax:

```
int ML_Operator_PrintSparsity(ML_Operator* Op, char* title,
                              char* FileName, int PrintDecomposition,
                              int NumPDEEqns);
```

`Op` is a pointer to the ML_Operator, `title` is a character array that will be inserted in the postscript file, `FileName` a valid file name. If `PrintDecomposition` is set to `ML_YES`, the postscript file will contain horizontal an vertical lines, corresponding to the division of the rows and columns among the processors. Finally, `NumPDEEqns` defines the number of PDE equations. If greater than 1, only the sparsity pattern of the block matrix will be plotted.

Two example of output are reported in Figure 1.

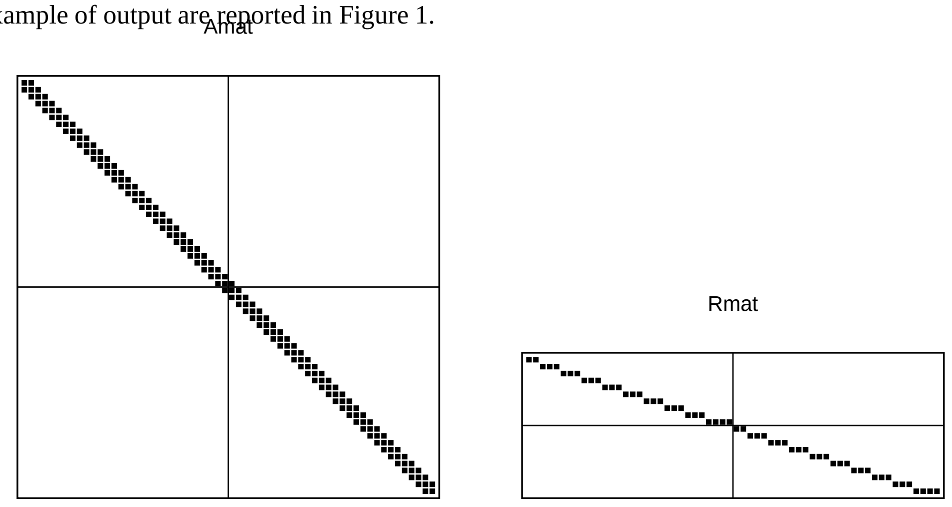

Figure 1. Sparsity pattern of two ML_Operators: the fi ne-level matrix (left) and the (non-smoothed) restriction operator from fi nest to coarsest level. The fi nest-level matrix has size 60, and corresponds to a 1D Laplacian on a Cartesian grid. The lines in the matrix show the division of rows and columns among the (two) processors.

13 OpendDX Visualization Capabilities

ML supports limited capabilities for the visualization of and statistical information for aggregates, with an interface to OpenDX. Currently, only `Uncoupled, METIS` and `ParMETIS` aggregation routines can dump files in OpenDX format.

The procedure to create the OpenDX input files is as follows:

1. Add the following line after the creation of the ML_Aggregate object

   ```
   ML_Aggregate_VizAndStats_Setup(ag, MaxMgLevels );
   ```

   where `MaxMgLevels` is the maximum number of levels (this is the same value used to create the ML object).

2. Create the multilevel hierarchy;

3. Write OpenDX file using the instruction

   ```
   ML_Aggregate_VizAndStats_Compute(ml, ag, MaxMgLevels, x, y, z,
                                    option, filename);
   ```

where `ml` is the ML object, `ag` the ML_Aggregation object, and `x,y,z` are double vectors, whose size equals the number of local nodes in the fine grid, containing the coordinates of fine grids nodes. `option` is an integer value defined so that:

- option = 1 : solution of 1D problem (`y` and `z` can be `NULL`);
- option = 2 : solution of 2D problems (`z` can be `NULL`);
- option = 3 : solution of 3D problems.

Processor X will write its own file, `filename_levelY_procX`, where `Y` is the level. `filename` can be set to `NULL` (default value of `.graph` will be used in this case).

Note that in AMG there is no mesh associated with coarser levels. Therefore ML_Aggregate_VizAndStats_Compute needs to assign a set of coordinates to each aggregate. This is done by computing the center of gravity of each aggregate (starting from the fine grid and finishing at the coarsest level).

`ML_Aggregate_VizAndStats_Compute` will also write statistical information to the screen.

4. Deallocate memory using

```
ML_Aggregate_VizAndStats_Clean( ag, MaxMgLevels ).
```

At this point, one should copy file `viz_aggre.net` and `viz_aggre.cfg` (located in `$ML_HOME/util/`) in the directory where the output files are located, and run OpendDX with the instruction

```
% dx -edit viz_aggre.net
```

Other instructions are reported in file `$ML_HOME/util/viz_aggre.README`. An example of code can be found in file `$ML_HOME/examples/ml_aztec_simple_METIS.c`.

Part V

Function Reference

14 ML Functions

Prototype

```
int AZ_ML_Set_Amat(ML *ml_object, int k, int isize, int osize, AZ_MATRIX *Amat,
                   int *proc_config)
```

Description

Create an ML matrix view of an existing **Aztec** matrix and store it within the 'ml_object' context.

Parameters

ml_object	On input, ML object pointer (see ML_Create). On output, the discretization matrix of level k is the same as given by Amat.
k	On input, indicates level within ml_object hierarchy (should be between 0 and Nlevels†-1).
isize	On input, the number of local rows in the submatrix stored on this processor.
osize	On input, the number of columns in the local submatrix stored on this processor not including any columns associated with ghost unknowns.
Amat	On input, an **Aztec** data structure representing a matrix. See the **Aztec** User's Guide.
proc_config	On input, an **Aztec** data structure representing processor information. See the **Aztec** User's Guide.

Prototype

```
void AZ_set_ML_preconditioner(AZ_PRECOND **Precond, AZ_MATRIX *Amat,
                              ML *ml_object, int options[])
```

Description

Associate the multigrid V cycle method defined in ml_object with an **Aztec** preconditioner. Thus, when Precond and options are passed into the **Aztec** iterative solver, it will invoke the V cycle multigrid algorithm described by ml_object.

Parameters

Precond	On input, an **Aztec** data structure representing a preconditioner. On output, the multigrid V cycle method described by ml_object will be associated with this preconditioner. See the **Aztec** User's Guide.
Amat	On input, an **Aztec** data structure representing a matrix. See the **Aztec** User's Guide.
ml_object	On input, ML object pointer (see ML_Create) representing a V cycle multigrid method.
options	On input, an **Aztec** data structure representing user chosen options. On output, set appropriately for multigrid V cycle preconditioner.

Prototype

```
int ML_Aggregate_Create(ML_Aggregate **agg_object)
```

Description

Create an aggregate context (or handle). This instance will be used in all subsequent function invocations that set aggregation options.

Parameters

agg_object	On input, a pointer to a noninitialized ML_Aggregate object pointer. On output, points to an initialized ML_Aggregate object pointer.

Prototype

```
int ML_Aggregate_Destroy(ML_Aggregate **agg_object)
```

Description

Destroy the aggregate context, agg_object, and delete all memory allocated by ML in building and setting the aggregation options.

Parameters

agg_object	On input, aggregate object pointer (see ML_Aggregate_Create). On output, all memory allocated by ML and associated with this context is freed.

Prototype

```
int ML_Aggregate_Set_CoarsenScheme_Coupled(ML_Aggregate *agg_object)
```

Description

Set the aggregate coarsening scheme to be used as ‘coupled’.

Parameters

agg_object	On input, aggregate object pointer (see ML_Aggregate_Create). On output, the ‘coupled’ aggregation will be used for automatic coarsening.

Prototype

```
int ML_Aggregate_Set_CoarsenScheme_MIS(ML_Aggregate *agg_object)
```

Description

Set the aggregate coarsening scheme to be used as ‘MIS’.

Parameters

agg_object	On input, aggregate object pointer (see ML_Aggregate_Create). On output, the ‘MIS’ aggregation will be used for automatic coarsening.

Prototype

```
int ML_Aggregate_Set_CoarsenScheme_Uncoupled(ML_Aggregate *agg_object)
```

Description

Set the aggregate coarsening scheme to be used as ‘uncoupled’.

Parameters

agg_object	On input, aggregate object pointer (see ML_Aggregate_Create). On output, the ‘uncoupled’ aggregation will be used for automatic coarsening.

Prototype

```
int ML_Aggregate_Set_CoarsenScheme_METIS(ML_Aggregate *agg_object)
```

Description

Set the aggregate coarsening scheme to be used as ‘METIS’.

Parameters

agg_object	On input, aggregate object pointer (see ML_Aggregate_Create). On output, the ‘METIS’ aggregation will be used for automatic coarsening.

Prototype

```
int ML_Aggregate_Set_CoarsenScheme_ParMETIS(ML_Aggregate *agg_object)
```

Description

Set the aggregate coarsening scheme to be used as ‘ParMETIS’.

Parameters

agg_object	On input, aggregate object pointer (see ML_Aggregate_Create). On output, the 'ParMETIS' aggregation will be used for automatic coarsening.

Prototype

```
int ML_Aggregate_Set_DampingFactor( ML_Aggregate *ag, double factor)
```

Description

Set the damping factor used within smoothed aggregation. In particular, the interpolation operator will be generated by

$$P = (I - \frac{\omega}{\tilde{\rho}} A) P_t$$

where A is the discretation matrix, ω is the damping factor (default is $\frac{4}{3}$), ρ is an estimate of the spectral radius of A, and P_t are the seed vectors (tentative prolongator).

Parameters

agg_object	On input, aggregate object pointer (see ML_Aggregate_Create). On output, the damping factor is set to factor.
factor	On input, damping factor that will be associated with this aggregation object.

Prototype

```
int ML_Aggregate_Set_MaxCoarseSize( ML_Aggregate *agg_object, int size )
```

Description

Set the maximum coarsest mesh to 'size'. No further coarsening is performed if the total number of matrix equations is less than this 'size'.

Parameters

agg_object	On input, aggregate object pointer (see ML_Aggregate_Create). On output, the coarsest mesh size will be set.
size	On input, size indicating the maximum coarsest mesh size.

Prototype

```
int ML_Aggregate_Set_NullSpace(ML_Aggregate *agg_object, int num_PDE_eqns, int null_dim,
                                        double *null_vect, int leng)
```

Description

Set the seed vectors (rigid body mode vectors) to be used in smoothed aggregation. Also indicate the number of degrees of freedom (DOF) per node so that the aggregation algorithm can group them together.

Parameters

agg_object	On input, an ML_Aggregate object pointer created by invoking ML_Aggregate_Create. On output, the seed vectors and DOFs per node are set to null_vect and num_PDE_eqns respectively.
num_PDE_eqns	On input, indicates number of equations that should be grouped in blocks when performing the aggregation. This guarantees that different DOFs at a grid point remain within the same aggregate.
null_dim	On input, number of seed vectors that will be used when creating the smoothed aggregation grid transfer operator.
null_vect	On input, the seed vectors are given in sequence. Each processor gives only the local components residing on the processor. If null, default seed vectors are used.
leng	On input, the length of each seed vector.

Prototype

```
int ML_Aggregate_Set_SpectralNormScheme_Calc( ML_Aggregate *ag )
```

Description

Set the method to be used for estimating the spectral radius of A (the discretization matrix) to be conjugate gradient. This spectral radius estimate is used when smoothing the initial prolongation operator (see ML_Aggregate_Set_DampingFactor).

Parameters

agg_object	On input, aggregate object pointer (see ML_Aggregate_Create). On output, the spectral radius estimate will be determined by a conjugate gradient routine.

Prototype

```
int ML_Aggregate_Set_SpectralNormScheme_Anorm( ML_Aggregate *ag)
```

Description

Set the method to be used for estimating the spectral radius of A (the discretization matrix) to be the infinity norm. This spectral radius estimate is used when smoothing the initial prolongation operator (see ML_Aggregate_Set_DampingFactor).

Parameters

agg_object	On input, aggregate object pointer (see ML_Aggregate_Create). On output, the spectral radius estimate will be taken as the infinity norm of the matrix.

Prototype

```
int ML_Aggregate_Set_Threshold(ML_Aggregate *agg_object, double tolerance)
```

Description

Set the drop tolerance used when creating the matrix graph for aggregation. Entries in the matrix A are dropped when $|A(i,j)| \leq tol\_d * \sqrt{|A(i,i)A(j,j)|}$.

Parameters

agg_object	On input, an ML_Aggregate object pointer created by invoking ML_Aggregate_Create. On output, drop tolerance for creating the matrix graph is set.
tolerance	On input, value to be used for dropping matrix entries.

Prototype

```
int ML_Create(ML **ml_object, int Nlevels)
```

Description

Create an ML solver context (or handle). This ML instance will be used in all subsequent ML function invocations. The ML object has a notation of levels where different multigrid operators corresponding to different grid levels are stored.

Parameters

ml_object	On input, a pointer to a noninitialized ML object pointer. On output, points to an initialized ML object pointer.
Nlevels	Maximum number of multigrid levels within this ML object.

Prototype

```
int ML_Destroy(ML **ml_object)
```

Description

Destroy the ML solver context, ml_object, and delete all memory allocated by ML in building and setting options.

Parameters

ml_object	On input, ML object pointer (see ML_Create). On output, all memory allocated by ML and associated with this context is freed.

Prototype

```
int ML_Gen_Blocks_Aggregates(ML_Aggregate *agg_object, int k, int *nblocks, int **block_list)
```

Description

Use aggregates to partition submatrix residing on local processor into blocks. These blocks can then be used within smoothers (see for example ML_Gen_Smoother_VBlockJacobi or ML_Gen_Smoother_VBlockSymGaussSeidel).

Parameters

ml_object	On input, ML object pointer (see ML_Create).
k	On input, indicates level within ml_object hierarchy where the aggregate information is found that defines partitioning.
nblocks	On output, indicates the number of partitions.
block_list	On output, equation i resides in the block_list[i]th partition.

Prototype

```
int ML_Gen_Blocks_Metis(ML *ml_object, int k, int *nblocks, int **block_list)
```

Description

Use Metis to partition submatrix residing on local processor into blocks. These blocks can then be used within smoothers (see for example ML_Gen_Smoother_VBlockJacobi or ML_Gen_Smoother_VBlockSymGaussSeidel).

Parameters

ml_object	On input, ML object pointer (see ML_Create).
k	On input, indicates level within ml_object hierarchy where the discretization matrix is found that will be partitioned.
nblocks	On input, indicates number of partitions desired on each processor. On output, indicates the number of partitions obtained.

block_list	On output, equation i resides in the block_list[i]th partition.

Prototype

```
int ML_Gen_CoarseSolverSuperLU(ML *ml_object, int k)
```

Description

Use SuperLU for the multigrid coarse grid solver on level k within ml_object and perform any initialization that is necessary.

Parameters

ml_object	On input, ML object pointer (see ML_Create). On output, the coarse grid solver of level k is set to use SuperLU.
k	On input, indicates level within ml_object hierarchy (must be the coarsest level in the multigrid hierarchy).

Prototype

```
int ML_Gen_MGHierarchy_UsingAggregation(ML *ml_object, int start, int inc_or_dec,
                                         ML_Aggregate *agg_object)
```

Description

Generate a multigrid hierarchy via the method of smoothed aggregation. This hierarchy includes a series of grid transfer operators as well as coarse grid approximations to the fine grid discretization operator. On completion, return the total number of multigrid levels in the newly created hiearchy.

Parameters

ml_object	On input, ML object pointer (see ML_Create). On output, coarse levels are filled with grid transfer operators and coarse grid discretizations corresponding to a multigrid hierarchy.

start	On input, indicates multigrid level within ml_object where the fine grid discretization is stored.
inc_or_dec	On input, ML_INCREMENT or ML_DECREMENT. Normally, set to ML_INCREMENT meaning that the newly created multigrid operators should be stored in the multigrid levels: start, start+1, start+2, start+3, etc. If Set to ML_DECREMENT, multigrid operators are stored in start, start-1, start-2, etc.
agg_object	On input, an initialized aggregation object defining options to the generation of grid transfer operators. If set to NULL, default values are used for all aggregation options. See ML_Aggregate_Create.

Prototype

```
int ML_Gen_SmootherAmesos(ML *ml_object, int k, int AmesosSolver,
                          int MaxProcs)
```

Description

Use Amesos interface to direct solvers for the multigrid coarse grid solver on level k within ml_object and perform any initialization that is necessary.

Parameters

ml_object	On input, ML object pointer (see ML_Create). On output, the coarse grid solver of level k is set to use Amesos.
k	On input, indicates level within ml_object hierarchy (must be the coarsest level in the multigrid hierarchy).
AmesosSolver	On input, indicates the direct solver library to use in the coarse solution. It can be: ML_AMESOS_UMFPACK, ML_AMESOS_KLU, ML_AMESOS_SUPERLUDIST, ML_AMESOS_MUMPS, ML_AMESOS_SCALAPACK.
MaxProcs	On input, indicates maximum number of processors to use in the coarse solution (only for ML_AMESOS_SUPERLUDIST).

Prototype

```
void ML_Gen_SmootherAztec(ML *ml_object, int k, int options[], double params[],
                          int proc_config[], double status[], int N_iterations,
                          int pre_or_post, void (*prec_fun)(double *, int *, int *,
                          double *, AZ_MATRIX *, AZ_PRECOND *))
```

Description

Set the smoother (either pre or post as indicated by pre_or_post) at level k within the multigrid solver context to invoke **Aztec** . The specific **Aztec** scheme is given by the **Aztec** arrays: options, params, proc_config, and status and **Aztec** preconditioning function: prec_function.

Parameters

ml_object	On input, ML object pointer (see ML_Create). On output, a smoother function is associated within ml_object at level k.
k	On input, indicates where the smoother function pointer will be stored within the multigrid hierarchy.
options, params proc_config, status	On input, **Aztec** arrays that determine the **Aztec** scheme and are used for **Aztec** to return information. See the **Aztec** User's Guide.
N_iterations	On input, maximum **Aztec** iterations within a single smoother invocation. When set to `AZ_ONLY_PRECONDITIONER`, only one iteration of the preconditioner is used without an outer Krylov method.
pre_or_post	On input, ML_PRESMOOTHER or ML_POSTSMOOTHER indicating whether the smoother should be performed before or after the coarse grid correction.
prec_fun	On input, **Aztec** preconditioning function indicating what preconditioner will be used within **Aztec** . Normally, this is set to AZ_precondition. See the **Aztec** User's Guide.

Prototype

```
int ML_Gen_Smoother_BlockGaussSeidel(ML *ml_object, int k, int pre_or_post, int ntimes,
                                     double omega, int blocksize)
```

Description

Set the multigrid smoother for level k of ml_object and perform any initialization that is necessary. When using block Gauss Seidel, the total number of equations must be a multiple of blocksize. Each consecutive group of blocksize unknowns is grouped into a block and a block Gauss Seidel algorithm is applied.

Parameters

ml_object	On input, ML object pointer (see ML_Create). On output, the pre or post smoother of level k is set to block Gauss Seidel.
k	On input, indicates level within ml_object hierarchy (should be between 0 and Nlevels†-1). ML_ALL_LEVELS sets the smoothing on all levels in ml_object.
pre_or_post	On input, ML_PRESMOOTHER or ML_POSTSMOOTHER indicating whether the pre or post smoother is to be set.
ntimes	On input, sets the number of block Gauss Seidel iterations that will be performed.
omega	On input, sets the damping parameter to be used during this block Gauss Seidel smoothing.
blocksize	On input, sets the size of the blocks to be used during block Gauss Seidel smoothing.

Prototype

```
int ML_Gen_Smoother_GaussSeidel(ML *ml_object, int k, int pre_or_post, int ntimes,
                                 double omega)
```

Description

Set the multigrid smoother for level k of ml_object and perform any initialization that is necessary.

Parameters

ml_object	On input, ML object pointer (see ML_Create). On output, the pre or post smoother of level k is set to Gauss Seidel.
k	On input, indicates level within ml_object hierarchy (should be between 0 and Nlevels†-1). ML_ALL_LEVELS sets the smoothing on all levels in ml_object.
pre_or_post	On input, ML_PRESMOOTHER or ML_POSTSMOOTHER indicating whether the pre or post smoother is to be set.
ntimes	On input, sets the number of Gauss Seidel iterations that will be performed.
omega	On input, sets the damping parameter to be used during this Gauss Seidel smoothing.

Prototype

```
int ML_Gen_Smoother_Jacobi(ML *ml_object, int k, int pre_or_post, int ntimes,
                           double omega)
```

Description

Set the multigrid smoother for level k of ml_object and perform any initialization that is necessary.

Parameters

ml_object	On input, ML object pointer (see ML_Create). On output, the pre or post smoother of level k is set to Jacobi.
k	On input, indicates level within ml_object hierarchy (should be between 0 and Nlevels†-1). ML_ALL_LEVELS sets the smoothing on all levels in ml_object.
pre_or_post	On input, ML_PRESMOOTHER or ML_POSTSMOOTHER indicating whether the pre or post smoother is to be set.
ntimes	On input, sets the number of Jacobi iterations that will be performed.
omega	On input, sets the damping parameter to be used during this Jacobi smoothing. ML_DEFAULT sets it to .5

Prototype

```
int ML_Gen_Smoother_SymGaussSeidel(ML *ml_object, int k, int pre_or_post, int ntimes,
                                   double omega)
```

Description

Set the multigrid smoother for level k of ml_object and perform any initialization that is necessary.

Parameters

ml_object	On input, ML object pointer (see ML_Create). On output, the pre or post smoother of level k is set to symmetric Gauss Seidel.
k	On input, indicates level within ml_object hierarchy (should be between 0 and Nlevels†-1). ML_ALL_LEVELS sets the smoothing on all levels in ml_object.

pre_or_post	On input, ML_PRESMOOTHER or ML_POSTSMOOTHER indicating whether the pre or post smoother is to be set.
ntimes	On input, sets the number of symmetric Gauss Seidel iterations that will be performed.
omega	On input, sets the damping parameter to be used during this symmetric Gauss Seidel smoothing.

Prototype

```
int ML_Gen_Smoother_VBlockJacobi(ML *ml_object, int k, int pre_or_post, int ntimes,
                                 double omega, int nBlocks, int *blockIndices)
```

Description

Set the multigrid smoother for level k of ml_object and perform any initialization that is necessary. A block Jacobi smoothing algorithm will be used where the size of the blocks can vary and is given by nBlocks and blockIndices (see ML_Gen_Blocks_Aggregates and ML_Gen_Blocks_Metis).

Parameters

ml_object	On input, ML object pointer (see ML_Create). On output, the pre or post smoother of level k is set to variable block Jacobi.
k	On input, indicates level within ml_object hierarchy (should be between 0 and Nlevels†-1). ML_ALL_LEVELS sets the smoothing on all levels in ml_object.
pre_or_post	On input, ML_PRESMOOTHER or ML_POSTSMOOTHER indicating whether the pre or post smoother is to be set.
ntimes	On input, sets the number of block Jacobi iterations that will be performed.
omega	On input, sets the damping parameter to be used during this block Jacobi smoothing.
nBlocks	On input, indicates the total number of block equations in matrix.
blockIndices	On input, blockIndices[i] indicates block to which ith element belongs.

Prototype

```
int ML_Gen_Smoother_VBlockSymGaussSeidel(ML *ml_object, int k, int pre_or_post,
                                         int ntimes, double omega, int nBlocks,
                                         int *blockIndices)
```

Description

Set the multigrid smoother for level k of ml_object and perform any initialization that is necessary. A block Gauss Seidel smoothing algorithm will be used where the size of the blocks can vary and is given by nBlocks and blockIndices (see ML_Gen_Blocks_Aggregates and ML_Gen_Blocks_Metis).

Parameters

ml_object	On input, ML object pointer (see ML_Create). On output, the pre or post smoother of level k is set to variable block symmetric Gauss Seidel.
k	On input, indicates level within ml_object hierarchy (should be between 0 and Nlevels†-1). ML_ALL_LEVELS sets the smoothing on all levels in ml_object.
pre_or_post	On input, ML_PRESMOOTHER or ML_POSTSMOOTHER indicating whether the pre or post smoother is to be set.
ntimes	On input, sets the number of block symmetric Gauss Seidel iterations that will be performed.
omega	On input, sets the damping parameter to be used during this block symmetric Gauss Seidel smoothing.
nBlocks	On input, indicates the total number of block equations in matrix.
blockIndices	On input, blockIndices[i] indicates block to which ith element belongs.

Prototype

```
int ML_Gen_Solver(ML *ml_object, int scheme, int finest_level, int coarsest_level)
```

Description

Initialize the ML solver context, ml_object, so that it is ready to be used in a solve. ML_Gen_Solver should be called after the multigrid cycle is fully specified but before ML_Iterate or ML_Solve_MGV is invoked.

Parameters

ml_object	On input, ML object pointer (see ML_Create). On output, all necessary initialization is completed.
scheme	On input, must be set to ML_MGV indicating a multigrid V cycle is used.
finest_level	On input, indicates the location within ml_object where the finest level is stored. Normally, this is '0'.
coarsest_level	On input, indicates location within ml_object where the coarsest grid is stored. When doing smoothed aggregation, this can be determined using the total number of multigrid levels returned by ML_Gen_MGHierarchy_UsingAggregation.

Prototype

```
int ML_Get_Amatrix(ML *ml_object, int k, ML_Operator **matrix)
```

Description

Set *matrix to point to the discretization matrix associated at level k within the multigrid solver context ml_object. This pointer can then be passed into functions like: ML_Operator_Apply, ML_Operator_Get_Diag, and ML_Operator_Getrow.

Parameters

ml_object	On input, ML object pointer (see ML_Create).
k	On input, indicates which level within the multigrid hierarchy should be accessed.
matrix	On output, *matrix points to the discretization matrix at level k within the multigrid hierarchy. This pointer can then be passed into the functions ML_Operator_Apply, ML_Operator_Get_Diag, and ML_Operator_Getrow.

Prototype

```
void * ML_Get_MyGetrowData(ML_Operator *Amat)
```

Description

Returns the user specific data pointer associated with the ML_Operator given by Amat. This function is normally employed when users write their own matrix getrow function and they need to get back the pointer that was given with ML_Init_Amatrix.

Parameters

Amat	On input, points to matrix for which we seek the internal data pointer.

Prototype

```
void * ML_Get_MyMatvecData(ML_Operator *Amat)
```

Description

Returns the user specific data pointer associated with the ML_Operator given by Amat. This function is normally employed when users write their own matrix-vector product function and they need to get back the pointer that was given with ML_Init_Amatrix.

Parameters

Amat	On input, points to matrix for which we seek the internal data pointer.

Prototype

```
void * ML_Get_MySmootherData(ML_Smoother *Smoother)
```

Description

Returns the user specific data pointer associated with the ML_Smoother object given by Smoother. This function is normally employed when users write their own smoother function and they need to get back the pointer that was given with ML_Set_Smoother.

Parameters

Smoother	On input, points to the smoother for which we seek the internal data pointer.

Prototype

```
int ML_Init_Amatrix(ML *ml_object, int k, int ilen, int olen, void *data)
```

Description

Set the size information for the discretization matrix associated at level k within ml_object. Additionally, associate a data pointer that can be used when writing matrix-vector product and matrix getrow functions.

Parameters

ml_object	On input, ML object pointer (see ML_Create). On output, size information is associated with the discretization matrix at level k.
k	On input, indicates where discretization size information will be stored within the multigrid hierarchy.
ilen	On input, the number of local rows in the submatrix stored on this processor.
olen	On input, the number of columns in the local submatrix stored on this processor not including any columns associated with ghost unknowns.
data	On input, a data pointer that will be associated with the discretization matrix and could be used for matrix-vector product and matrix getrow functions.

Prototype

```
int ML_Iterate(ML *ml_object, double *sol, double *rhs)
```

Description

Iterate until convergence to solve the linear system using the multigrid V cycle defined within ml_object.

Parameters

ml_object	On input, ML object pointer (see ML_Create).
sol	On input, a vector containing the initial guess for the linear system contained in ml_object. On output, the solution obtained by performing repeated multigrid V cycles.
rhs	On input, a vector contain the right hand side for the linear system contained in ml_object.

Prototype

```
int ML_Operator_Apply(ML_Operator *A, int in_length, double p[], int out_length,
                      double ap[])
```

Description

Invoke a matrix-vector product using the ML_Operator A. That is perform $ap = A * p$. Any communication or ghost variables work needed for this operation is also performed.

Parameters

A	On input, an ML_Operator (see ML_Get_Amatrix).
in_length	On input, length of vector p (not including ghost variable space).
p	On input, vector which will be multiplied by A.
out_length	On input, length of vector ap.
ap	On output, vector containing result of $A * p$.

Prototype

```
int ML_Operator_Get_Diag(ML_Operator *A, int length, double **diag)
```

Description

Get the diagonal of the ML_Operator `A` (which is assumed to be square).

Parameters

A	On input, an ML_Operator (see ML_Get_Amatrix).
length	On input, number of diagonal elements wanted.
diag	On output, sets a pointer to an array containing the diagonal elements. NOTE: this is not a copy but in fact a pointer into an ML data structure. Thus, this array should not be freed.

Prototype

```
int ML_Operator_Getrow(ML_Operator *A, int N_requested_rows, int requested_rows[],
                       int allocated_space, int columns[], double values[],
                       int row_lengths[])
```

Description

Get a row (or several rows) from the ML_Operator `A`. If there is not enough space in `columns` and `values` to hold the nonzero information, this routine returns a '0'. Otherwise, a '1' is returned.

Parameters

A	On input, an ML_Operator (see ML_Get_Amatrix).
N_requested_rows	On input, number of matrix rows for which information is returned.
requested_rows	On input, specific rows for which information will be returned.
allocated_space	On input, length of `columns` and `values`.
columns	On output, the column numbers of each nonzero within each row requested in `requested_rows` (where column numbers associated with `requested_rows[i]` appear before column numbers associated with `requested_rows[j]` with `i` $<$ `j`).
values	On output, the nonzero values of each nonzero within each row requested in `requested_rows` (where nonzero values associated with `requested_rows[i]` appear before nonzero values associated with `requested_rows[j]` with `i` $<$ `j`).

row_lengths	On output, `row_lengths[i]` indicates the number of nonzeros in row `i`.

Prototype

```
int ML_Set_Amatrix_Getrow(ML *ml_object, int k, int (*getrow)(ML_Operator *, int , int* , int,
                          int*, double* , int*), int (*comm )(double *vec, void *data),
                          int comm_vec_leng)
```

Description

Set the matrix getrow function for the discretization matrix associated at level k within the multigrid solver context ml_object.

Parameters

ml_object	On input, ML object pointer (see ML_Create). On output, matrix getrow function is associated with the discretization matrix at level k.
k	On input, indicates where the matrix getrow function pointer will be stored within the multigrid hierarchy.
getrow	On input, a function pointer to the user-defined matrix getrow function.
comm	On input, a function pointer to the user-defined communication function.

Prototype

```
int ML_Set_Amatrix_Matvec(ML *ml_object, int k, int (*matvec)(ML_Operator *, int, double *,
                          int, double *))
```

Description

Set the matrix-vector product function for the discretization matrix associated at level k within the multigrid solver context ml_object.

Parameters

ml_object	On input, ML object pointer (see ML_Create). On output, matrix-vector product function is associated with the discretization matrix at level k.
k	On input, indicates where the matrix-vector product function pointer is stored within the multigrid hierarchy.
matvec	On input, a function pointer to the user-defined matrix-vector product function.

.

Prototype

```
int ML_Set_ResidualOutputFrequency(ML *ml_object, int output_freq)
```

Description

Set the output frequency of residual information. ML_Iterate prints the two norm of the residual every output_freq iterations.

Parameters

ml_object	On input, ML object pointer (see ML_Create). On output, residual printing frequency is set.
output_freq	On input, value to use for printing frequency.

Prototype

```
int ML_Set_Smoother(ML *ml_object, int k , int pre_or_post, void *data,
                    int (*func)(ML_Smoother *, int, double *, int, double *), char *label)
```

Description

Set the smoother (either pre or post as indicated by pre_or_post) at level k within the multigrid solver context to invoke the user-defined function ‘func’ and pass in the data pointer ‘data’ via ML_Get_MySmootherData.

Parameters

ml_object	On input, ML object pointer (see ML_Create). On output, a smoother function is associated within ml_object at level k.
k	On input, indicates where the smoother function pointer will be stored within the multigrid hierarchy.
pre_or_post	On input, ML_PRESMOOTHER or ML_POSTSMOOTHER indicating whether the smoother should be performed before or after the coarse grid correction.
data	On input, a data pointer that will be passed into the user-defined function 'func'.
func	On input, smoothing function to be used at level k when performing a multigrid V cycle. The specific signature and details of this function are given in the Users Guide.
label	On input, a character string to be associated with Smoother. This string is printed by some routines when identifying the method.

Prototype

```
int ML_Set_Tolerance(ML *ml_object, double tolerance)
```

Description

Set the convergence criteria for ML_Iterate. Convergence is declared when the 2-norm of the residual is reduced by 'tolerance' over the initial residual. This means that if the initial residual is quite small (i.e. the initial guess corresponds quite closely with the true solution), ML_Iterate might continue to iterate without recognizing that the solution can not be improved due to round-off error. Note: the residual is always computed after performing presmoothing on the finest level (as opposed to at the beginning or end of the iteration). Thus, the true residual should be a little bit better than the one used by ML.

Parameters

ml_object	On input, ML object pointer (see ML_Create). On output, tolerance is set for convergence of ML_Iterate.
tolerance	On input, value to use for convergence tolerance.

Prototype

```
int ML_Solve_MGV(ML *ml_object, double *din, double *dout)
```

Description

Perform one multigrid V cycle iteration to the solve linear system defined within ml_object.

Parameters

ml_object	On input, ML object pointer (see ML_Create).
din	On input, the right hand side vector to be used when performing multigrid.
dout	On output, an approximate solution obtained after one multigrid V cycle.

[†]Nlevels refers to the argument given with `ML_Create`.

References

[Frea] Free Software Foundation. Autoconf Home Page. http://www.gnu.org/software/autoconf.

[Freb] Free Software Foundation. Automake Home Page. http://www.gnu.org/software/automake.

[HWH03] Michael A. Heroux, James M. Willenbring, and Robert Heaphy. Trilinos Developers Guide. Technical Report SAND2003-1898, Sandia National Laboratories, 2003.

[MAH03] P. .M. Sexton M. A. Heroux. Epetra developers coding guidelines, 2003.

[SHT04] M. Sala, J.J. Hu, and R.S. Tuminaro. ML3.0 Smoothed Aggregation User's Guide. Technical Report SAND2004-2195, Sandia National Laboratories, Albq, NM, 2004.